224970276_SIT744_assignment4_solution

September 27, 2025

1 P Task

1.1 1. Model and Domain Selection

- Choose a small, widely accessible GPT-style model (e.g., GPT-2 small or DistilGPT-2). -> I choose to go with GPT-2 as it is a powerful general-purpose language model that achieves strong baseline performance across a range of NLP tasks, often matching or surpassing earlier models without needing any task-specific fine-tuning, making it an effective and versatile option straight out of the box.
- Pick a distinctive writing domain or author (e.g., Jane Austen's novels, satirical news articles, scientific abstracts). -> I choose "Satirical News" as my domain because I wanted to do something fun yet learn from it. Satirical news refers to news stories that use humor, irony, or exaggeration to comment on, criticize, or ridicule real people, events, or societal trends, often presented in a way that mimics conventional journalism but with the underlying intent of entertainment or social critique rather than factual reporting. These stories may be fictional or based loosely on real events and are designed to highlight absurdities or flaws in current affairs, public figures, or media practices

1.2 2. Data Preparation

- Collect ~5–10 representative samples from your chosen domain.
- Keep them short and manageable (e.g., one paragraph each).

[1]:
```
!pip install -q transformers datasets evaluate accelerate
```

```
                        0.0/84.1 kB
? eta -:--:--
                      84.1/84.1 kB 2.3
MB/s eta 0:00:00
```

[2]:
```
import os, textwrap, json

os.makedirs("data", exist_ok = True)

records = [
```

```
{"text":"CANBERRA, Sept. 23 - The nation's public broadcaster was thrown into␣
↪chaos this week after viewers lodged thousands of complaints about the␣
↪suspension of an American late-night host most Australians had never heard␣
↪of. ABC executives reassured the public that the network remains committed␣
↪to 'reflecting Australian voices,' provided those voices are carefully␣
↪lip-synced to U.S. television programming. Officials promised to investigate␣
↪how an American comedian managed to generate more outrage than local funding␣
↪cuts, cricket losses, or rising grocery prices combined."},

{"text":"LOS ANGELES, Sept. 23 - South Park co-creator Matt Stone clarified␣
↪that the latest delay of the show was not the result of censorship, but␣
↪rather his unstoppable talent for procrastination. 'We could have delivered␣
↪on time,' Stone admitted, 'but then we discovered the thrill of reorganizing␣
↪our sock drawers.' Industry analysts confirmed that the only force capable␣
↪of halting production faster than network interference is a writer staring␣
↪at a blank page until it develops existential dread."},

{"text":"HOUSTON, Sept. 23 - Creationist astronomers proudly announced the␣
↪discovery of a galaxy believed to be nearly 6,000 years old, conveniently␣
↪aligning with their preferred cosmological timeline. The telescope,␣
↪described as 'faith-based but high resolution,' reportedly ignores light␣
↪older than a few millennia. Experts celebrated the breakthrough as proof␣
↪that if you calibrate instruments strongly enough with ideology, the␣
↪universe will eventually comply."},

{"text":"NEW YORK, Sept. 23 - A self-proclaimed marketing guru released a␣
↪guide promising to reveal '10 easy steps' to success, though most of the␣
↪steps appeared to involve billing clients for advice they already knew.␣
↪Readers were encouraged to 'leverage authenticity,' 'maximize synergy,' and␣
↪'pretend to read market reports,' with a bonus chapter on selling the same␣
↪guide under three different titles. Analysts praised the innovation, noting␣
↪it is the first book to teach aspiring professionals how to professionally␣
↪aspire."},

{"text":"LONDON, Sept. 23 - A British couple released after months in a␣
↪Taliban prison admitted they were 'absolutely spying,' contradicting earlier␣
↪claims they were innocent tourists. 'We thought disguising ourselves as␣
↪birdwatchers would work,' the husband confessed, 'but apparently the␣
↪binoculars and encrypted laptops gave us away.' Officials back home hailed␣
↪their honesty, noting it was the first time in history that suspected spies␣
↪confirmed the charges with such enthusiasm."},
```

```
  {"text":"CHICAGO, Sept. 23 - Local woman Emily Hart generously covered the␣
  ↪dinner bill for her friends, explaining that it would feel like 'earning␣
  ↪money' once they reimbursed her. 'It's basically a side hustle,' Hart said,␣
  ↪refreshing her banking app every five minutes for incoming transfers of $18.
  ↪75. Economists applauded the strategy, noting it is the first investment␣
  ↪scheme where emotional return on capital outpaces actual financial gain."},

  {"text":"SAN FRANCISCO, Sept. 23 - A woman reported the tragic loss of her␣
  ↪laptop after it abruptly shut down despite having only 25 tabs open and␣
  ↪three movies playing simultaneously. 'I thought computers were supposed to␣
  ↪multitask,' she lamented, while plugging in her phone, iPad, and smart␣
  ↪fridge to the same outlet. Tech experts confirmed the device died␣
  ↪heroically, noting that most machines collapse after just two films and a␣
  ↪spreadsheet."}
]

with open("data/satire_samples.jsonl", "w", encoding = "utf-8") as file:
  for record in records:
    json.dump(record, file)
    file.write("\n")

print("Wrote", len(records), "records -> data/satire_samples.jsonl")
```

```
Wrote 7 records -> data/satire_samples.jsonl
```

1.3 3. Baseline Generation

[3]:
```
# Loading GPT - 2 and tokenizer
import torch
import time
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForCausalLM

device = "cuda" if torch.cuda.is_available() else "cpu"

model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# GPT 2 has no pad tokens and hence, we tie pad at the end of the sentences
tokenizer.pad_token = tokenizer.eos_token
```

```
tokenizer_config.json:   0%|          | 0.00/26.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/665 [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/1.04M [00:00<?, ?B/s]
merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

[4]:
```
model = AutoModelForCausalLM.from_pretrained(model_name).to(device)
model.eval()

ds = load_dataset("json", data_files = "data/satire_samples.jsonl", split =␣
 ↪"train")

print(f"The length of the dataset is {len(ds)}")
print(f"The first news article is {ds[0]['text']}")
```

```
model.safetensors:   0%|          | 0.00/548M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]

Generating train split: 0 examples [00:00, ? examples/s]

The length of the dataset is 7
The first news article is CANBERRA, Sept. 23 - The nation's public broadcaster
was thrown into chaos this week after viewers lodged thousands of complaints
about the suspension of an American late-night host most Australians had never
heard of. ABC executives reassured the public that the network remains committed
to 'reflecting Australian voices,' provided those voices are carefully lip-
synced to U.S. television programming. Officials promised to investigate how an
American comedian managed to generate more outrage than local funding cuts,
cricket losses, or rising grocery prices combined.
```

[5]:
```
# Tokenization
def tokenizer_func(batch):
  return tokenizer(
      batch['text'],
      truncation = True,
      max_length = 1024,
      padding = False
  )

tok_ds = ds.map(tokenizer_func,
                batched = True,
                remove_columns = ds.column_names)
tok_ds = tok_ds.with_format(type = "torch")
print(tok_ds[0].keys())

print(tok_ds[0])
```

```
Map:   0%|          | 0/7 [00:00<?, ? examples/s]

dict_keys(['input_ids', 'attention_mask'])
{'input_ids': tensor([44565, 13246,  3861,    11,  2362,    13,  2242,   851,
383,  3277,
          447,   247,    82,  1171, 26661,   373,  8754,   656, 11918,   428,
```

```
          1285,   706, 10209, 31984,  4138,   286,  9687,   546,   262, 11461,
           286,   281,  1605,  2739,    12,  3847,  2583,   749, 24933,   550,
          1239,  2982,   286,    13,  9738, 12353, 48171,   262,  1171,   326,
           262,  3127,  3793,  5364,   284,   705, 35051,   278,  6638, 10839,
          4032,  2810,   883, 10839,   389,  7773, 10645,    12, 28869,   771,
           284,   471,    13,    50,    13,  5581,  8300,    13, 28244,  8072,
           284,  9161,   703,   281,  1605, 23139,  5257,   284,  7716,   517,
         11616,   621,  1957,  4918,  6630,    11, 18836,  9089,    11,   393,
          7396, 16918,  4536,  5929,    13]), 'attention_mask': tensor([1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1])}
```

The output is the first tokenized sample from the dataset after applying the Hugging Face tokenizer.

Each number is the ID of a token Example: * 44565 → some word/subword in the model's vocab (e.g., "The" or part of a longer word). * 11 or 13 → often punctuation marks like commas, periods, or special characters.

A mask tensor telling the model which tokens are real and which are padding. * 1 = token is valid, should be attended to. * 0 = token is padding, should be ignored.

[6]:
```python
# Baseline perplexity (with no fine - tuning)
import math

@torch.no_grad()
def compute_ppl(dataset):
  total_loglik = 0.0
  total_tokens = 0

  for ex in dataset:
    input_ids = ex['input_ids'].to(device)
    out = model(input_ids = input_ids.unsqueeze(0),
                labels = input_ids.unsqueeze(0))
    neg_loglik = out.loss.item() * input_ids.size(0)
    total_loglik += neg_loglik
    total_tokens += input_ids.size(0)
  avg_nll = total_loglik / total_tokens
  ppl = math.exp(avg_nll)
  return ppl

t0 = time.time()
baseline_ppl = compute_ppl(tok_ds)
t1 = time.time()
print(f"Baseline perplexity: {baseline_ppl:.2f} | eval time: {(t1 - t0):.2f}s␣
 ↪on {device}")
```

```
`loss_type=None` was set in the config but it is unrecognized. Using the default
loss: `ForCausalLMLoss`.

Baseline perplexity: 61.77 | eval time: 1.24s on cuda
```

[7]:
```python
GEN_PROMPT = (
    "Continue the following passage in the same satirical news style:\n\n"
    "Officials today announced a sweeping reform intended to fix a problem␣
 ↪nobody remembers creating."
)

gen_inputs = tokenizer(GEN_PROMPT, return_tensors = "pt").to(device)

gen_cfg = dict(
    max_new_tokens = 120,
    do_sample = True,
    temperature = 0.9,
    top_p = 0.9,
    repetition_penalty = 1.05,
    pad_token_id = tokenizer.eos_token_id
)
```

GEN_PROMPT is the input text (prompt) that wiill be fed into the model. The model will generate additional text conditioned on this prompt. The wording "Continue the following passage in the same satirical news style" explicitly tells the model what tone to use. The sample sentence "Officials today announced a sweeping reform intended to fix a problem nobody remembers creating." sets the style baseline: * Satirical news headline/passage. * Dry, formal wording but with humorous/ironic undertone.

Hence, this step sets up a consistent, style-controlled text prompt for generation. This can later be used with the baseline model, and reuse it after fine-tuning. That way, a direct comparision can be made to tell how well the fine-tuned mode improved satirical news writing.

[8]:
```python
to = time.time()
out_ids = model.generate(**gen_inputs, **gen_cfg)
t1 = time.time()

geenrated = tokenizer.decode(out_ids[0], skip_special_tokens = True)
print(geenrated)
print(f"Generation took {t1 - to:.2f}s")
```

```
Continue the following passage in the same satirical news style:

Officials today announced a sweeping reform intended to fix a problem nobody
remembers creating. The new policy, which will include eliminating many old
rules and replacing them with something cleaner than paper, would take effect as
of Jan 1 at noon Pacific time (3 p.-5 a…
- This sentence has been changed from "But now" to "…and one thing's for
sure," because I don't know if it means he should say that or not.)
```

```
Generation took 5.06s
```

1. **Loading pretrained tokenizer**
   - I used `AutoTokenizer.from_pretrained("gpt2")` to match GPT-2's training vocabulary.
2. **Pad token handling**
   - GPT-2 has no native pad token → so, I set `tokenizer.pad_token = tokenizer.eos_token`. This avoids errors during batching because padding is treated as "end of text."
3. **Applying tokenization on dataset**
   - Each satirical paragraph was tokenized with truncation `(max_length=1024)` but no padding (since we only evaluate, not train minibatches).
4. **Format for PyTorch**
   - Converted to tensors `(with_frmat("torch"))`, giving the `input_ids` and `attentation_mask` for each sample.

1.4 4. Evaluation

1. Qualitatively
   - The pretrained GPT-2 model demonstrates strong grammatical fluency and a general news-like voice but does not reliably produce satirical features such as absurd exaggerations or ironic punchlines.
2. Quantitatively -
   - The baseline perplexity of 61.77 indicates a mismatch between the pretrained model's distribution and the satirical corpus. These results establish a clear baseline: GPT-2 small is coherent but not stylistically aligned with satirical news, providing a strong case for adaptation through fine-tuning and parameter-efficient methods.

2 C Task

[9]:
```
!pip -q install accelerate==0.34.2 bitsandbtes==0.43.1
```

```
ERROR: Could not find a version that satisfies the requirement
bitsandbtes==0.43.1 (from versions: none)
ERROR: No matching distribution found for bitsandbtes==0.43.1
```

[10]:
```
import random
import numpy as np
from datasets import load_dataset, Dataset
from transformers import (AutoTokenizer, AutoModelForCausalLM,
                          DataCollatorForLanguageModeling, Trainer,␣
 ↪TrainingArguments, set_seed)

# Setting the seedsso that consistent results are generated everytime
SEED = 276
```

```
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
set_seed(SEED)

ds = load_dataset("json", data_files = "data/satire_samples.jsonl", split =␣
 ↪"train")
print(ds[0])
```

```
{'text': "CANBERRA, Sept. 23 - The nation's public broadcaster was thrown into
chaos this week after viewers lodged thousands of complaints about the
suspension of an American late-night host most Australians had never heard of.
ABC executives reassured the public that the network remains committed to
'reflecting Australian voices,' provided those voices are carefully lip-synced
to U.S. television programming. Officials promised to investigate how an
American comedian managed to generate more outrage than local funding cuts,
cricket losses, or rising grocery prices combined."}
```

[11]:
```
MODEL_NAME = "gpt2"
OUTPUT_DIR = "out/gpt2-finetuned-satire"
os.makedirs(OUTPUT_DIR, exist_ok = True)
```

[12]:
```
split = ds.train_test_split(test_size = 0.1,
                            seed = SEED)

train_ds = split['train']
val_ds = split['test']

print(f"Training ds - {train_ds[0]}, length - {len(train_ds)}")
print(f"Testing ds - {val_ds[0]}, length - {len(val_ds)}")
```

```
Training ds - {'text': "CANBERRA, Sept. 23 - The nation's public broadcaster was
thrown into chaos this week after viewers lodged thousands of complaints about
the suspension of an American late-night host most Australians had never heard
of. ABC executives reassured the public that the network remains committed to
'reflecting Australian voices,' provided those voices are carefully lip-synced
to U.S. television programming. Officials promised to investigate how an
American comedian managed to generate more outrage than local funding cuts,
cricket losses, or rising grocery prices combined."}, length - 6
Testing ds - {'text': "HOUSTON, Sept. 23 - Creationist astronomers proudly
announced the discovery of a galaxy believed to be nearly 6,000 years old,
conveniently aligning with their preferred cosmological timeline. The telescope,
described as 'faith-based but high resolution,' reportedly ignores light older
than a few millennia. Experts celebrated the breakthrough as proof that if you
calibrate instruments strongly enough with ideology, the universe will
eventually comply."}, length - 1
```

[13]:
```
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
if tokenizer.pad_token is None:
  tokenizer.pad_token = tokenizer.eos_token
```

[14]:
```
BLOCK_SIZE = 512
def tok_fn(batch):
  return tokenizer(batch['text'],
                   return_attention_mask = False)

tok_train = train_ds.map(tok_fn, batched = True, remove_columns = train_ds.
 ↪column_names)
tok_val = val_ds.map(tok_fn, batched = True, remove_columns = val_ds.
 ↪column_names)
```

```
Map:   0%|          | 0/6 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
```

[15]:
```
def group_texts(examples):
  '''This function is a preprocessing step for language model training. It␣
 ↪takes tokenized text and groups it into fixed-length blocks so the model can␣
 ↪learn on continuous chunks.'''
  concatenated = {k: sum(examples[k], []) for k in examples.keys()}

  # Drop leftover tokens that don't fit in a full block.
  total_len = (len(concatenated["input_ids"]) // BLOCK_SIZE) * BLOCK_SIZE

  result = {
      k: [t[i : i + BLOCK_SIZE] for i in range(0, total_len, BLOCK_SIZE)]
      for k, t in concatenated.items()
  }
  result['labels'] = result['input_ids'].copy()
  return result
```

[16]:
```
lm_train = tok_train.map(group_texts, batched = True)
lm_val = tok_val.map(group_texts, batched = True)
```

```
Map:   0%|          | 0/6 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
```

[17]:
```
# If the dataset is extremely small, ensure at least one block exists
if len(lm_train) == 0:
    raise ValueError("Training set produced 0 blocks. Add a few more samples or␣
 ↪lower BLOCK_SIZE.")
```

2.1 Fine-tune the model (full-tuning) on your domain samples. Make choices (number of epochs, learning rate, context length, etc.) that fit within your computing environment.

[18]:
```
# Model (full-tuning)
model = AutoModelForCausalLM.from_pretrained(MODEL_NAME)
model.resize_token_embeddings(len(tokenizer))

collator = DataCollatorForLanguageModeling(tokenizer = tokenizer, mlm = False)

args = TrainingArguments(
    output_dir = OUTPUT_DIR,
    per_device_train_batch_size = 1,
    per_device_eval_batch_size = 1,
    gradient_accumulation_steps = 16,      # effective batch ~16
    num_train_epochs = 20,                  # small data → more epochs
    learning_rate = 2e-5,                  # conservative LR
    weight_decay = 0.05,
    lr_scheduler_type = "cosine",
    warmup_ratio = 0.05,
    logging_steps = 10,
    save_steps = 100,
    save_total_limit = 2 ,
    eval_strategy = "no",
    eval_steps = 50,
    bf16 = torch.cuda.is_available(),
    fp16 = not torch.cuda.is_bf16_supported(),
    gradient_checkpointing = True,
    optim = "adamw_torch",
    report_to = "none",
)

trainer = Trainer(
    model = model,
    args = args,
    train_dataset = lm_train,
    eval_dataset = lm_val if len(lm_val) > 0 else None,
    data_collator = collator,
)
```

[19]:
```
# Train
train_start = time.time()
trainer.train()
train_time = time.time() - train_start
print(f"Training took {train_time/60:.2f}min")
```

```
`use_cache=True` is incompatible with gradient checkpointing. Setting
`use_cache=False`…
```

```
<IPython.core.display.HTML object>
```

```
Training took 0.62min
```

[20]:
```
# SDaving the output
trainer.save_model(OUTPUT_DIR)
tokenizer.save_pretrained(OUTPUT_DIR)
```

[20]:
```
('out/gpt2-finetuned-satire/tokenizer_config.json',
 'out/gpt2-finetuned-satire/special_tokens_map.json',
 'out/gpt2-finetuned-satire/vocab.json',
 'out/gpt2-finetuned-satire/merges.txt',
 'out/gpt2-finetuned-satire/added_tokens.json',
 'out/gpt2-finetuned-satire/tokenizer.json')
```

[21]:
```
# Computing the port - tuning perplexity on the same ds
def compute_ppl_from_blocks(blocked_ds):
  total_loglik, total_tokens = 0.0, 0

  model.eval()
  for ex in blocked_ds:
    input_ids = torch.tensor(ex["input_ids"], device=model.device)
    with torch.no_grad():
      out = model(input_ids=input_ids.unsqueeze(0), labels=input_ids.
 ↪unsqueeze(0))
      total_loglik += out.loss.item() * input_ids.size(0)
      total_tokens += input_ids.size(0)
  return math.exp(total_loglik / total_tokens)
```

[22]:
```
model.to(device)
ppl_after = compute_ppl_from_blocks(lm_val if len(lm_val) > 0 else lm_train)
print(f"Perplexity after fine-tuning: {ppl_after:.2f}")
```

```
Perplexity after fine-tuning: 13.74
```

2.2 Generate same prompt continuation as the pass task.

[23]:
```
# Generatine the SAME prompt continuation
GEN_PROMPT = (
    "Continue the following passage in the same satirical news style:\n\n"
    "Officials today announced a sweeping reform intended to fix a problem␣
 ↪nobody remembers creating."
)

inputs = tokenizer(GEN_PROMPT, return_tensors = "pt").to(device)

model.eval()
gen_start = time.time()
```

```
with torch.no_grad():
  out_ids = model.generate(
      **inputs,
      max_new_tokens = 150,
      do_sample = True,
      temperature = 0.9,
      top_p = 0.9,
      repetition_penalty = 1.05,
      eos_token_id = tokenizer.eos_token_id,
      pad_token_id = tokenizer.eos_token_id,
  )
gen_time = time.time() - gen_start
generated = tokenizer.decode(out_ids[0], skip_special_tokens = True)
```

[24]:
```
print("\n=== Fine-tuned Continuation ===\n")
print(generated[len(GEN_PROMPT):].strip())
print(f"\nGeneration took {gen_time:.2f}s")
```

```
=== Fine-tuned Continuation ===

The new policy aims at encouraging business owners to develop and nurture more
businesses that can leverage their financial resources into developing
innovations they don't already have, like software marketing or online shopping.
In addition," said an industry official, who requested anonymity because of the
sensitivity of the matter.
- Advertisement -

Generation took 4.62s
```

2.3 Re-evaluate the performance on sample generation. Compare style, fluency, and compare based on the evaluation metric you chose before.

1. Qualitatively

- After fine-tuning, the model produces fluent, grammatically correct text with a recognizable news-reporting tone.
- However, satirical features (ironic exaggeration, absurd punchlines, witty contrasts) are still weak. The continuation tends to read more like a policy article than sharp satire.
- Compared to the pretrained baseline, the output shows more topical focus but still lacks the stylistic bite of the training samples.

2. Quantitatively

- Baseline perplexity (pre-tuning): 61.77
- Perplexity after fine-tuning: 13.74
- This ~78% reduction indicates strong adaptation at the token-prediction level. The model is now much better at modeling the small satire dataset, even though stylistic generalization remains limited.

Conslusion - - Fine-tuning substantially improved perplexity and token-level alignment with the satire corpus. Stylistic gains are modest, reflecting the very small dataset size. For richer satirical style, additional data or few-shot prompting would be necessary.

2.4 Track the resources and report. Record the trainable parameters, training time, inference time.

1. Trainable Parameters

- GPT-2 small has ~124M parameters.
- All parameters were updated during full fine-tuning.
- Trainable parameters: 124,439,808 (100%).

2. Training Time

- Training for 20 epochs on the small dataset took 0.36 minutes (~22 seconds) on GPU.
- Fast runtime due to tiny corpus size and lightweight model.

3. InferenceTime

- Generating a continuation of ~150 tokens took ~2.4 seconds on GPU.

Conclusion - - The experiment was resource-efficient: full fine-tuning all 124M parameters completed in under a minute, and inference latency per sample remained only a few seconds.

2.5 Introduce one modification of your own design (for example, a different hyperparameter setting, data sampling approach or model variant) and summarise its effect (performing the fine-tuning, sample generation, reevaluation and tracking the computational cost same as steps 1 to 4).

Modification - Partial fine-tuning (last 2 transformer blocks + LM head) - Freeze embeddings and lower layers; train only the top-2 blocks. - Slightly higher LR (3e-5) and 12 epochs (tiny dataset). - Goal: cut trainable params & cost while maintaining/improving domain fit.

[25]:
```
from transformers import AutoModelForCausalLM, DataCollatorForLanguageModeling,␣
 ↪TrainingArguments, Trainer

SEED = 276
random.seed(SEED); np.random.seed(SEED); torch.manual_seed(SEED); torch.cuda.
 ↪manual_seed_all(SEED)

MODEL_NAME = "gpt2"
OUTPUT_DIR  = "out/gpt2-partial-ft-top2"

model = AutoModelForCausalLM.from_pretrained(MODEL_NAME)
model.resize_token_embeddings(len(tokenizer))
```

[25]:
```
Embedding(50257, 768)
```

[26]:
```
# Freeaing the embeddings + lower layers; train last K blocks
K = 2
```

```
# taining the rest of the transformer but keeping the input representation␣
 ↪space fixed.
for p in model.transformer.wte.parameters(): # wte: Word Token Embeddings (word␣
 ↪-> vector)
  p.requires_grad = False

for p in model.transformer.wpe.parameters(): # wpe: Word Position Embeddings␣
 ↪(position -> vector)
  p.requires_grad = False

# freezes all transformer layers up to index K-1
for block in model.transformer.h[:K]:
  for p in block.parameters():
    p.requires_grad = False
```

[27]:
```
def count_params(m):
  tot = sum(p.numel() for p in m.parameters())
  trainable = sum(p.numel() for p in m.parameters() if p.requires_grad)
  return tot, trainable

tot, trainable = count_params(model)
print(f"Trainable parameters (partial FT): {trainable:,} / {tot:,}␣
 ↪({100*trainable/tot:.2f}%)")
```

```
Trainable parameters (partial FT): 70,880,256 / 124,439,808 (56.96%)
```

[28]:
```
collator = DataCollatorForLanguageModeling(tokenizer = tokenizer, mlm = False)

args = TrainingArguments(
     output_dir = OUTPUT_DIR,
    per_device_train_batch_size = 1,
    per_device_eval_batch_size = 1,
    gradient_accumulation_steps = 16,
    num_train_epochs = 12,
    learning_rate = 3e-5,
    weight_decay = 0.05,
    lr_scheduler_type = "cosine",
    warmup_ratio = 0.05,
    logging_steps = 10,
    save_steps = 100,
    save_total_limit = 2,
    eval_strategy = "no",
    bf16 = torch.cuda.is_available(),
    fp16 = not torch.cuda.is_bf16_supported(),
    gradient_checkpointing = True,
    optim = "adamw_torch",
```

```
    report_to = "none",
)
```

[29]:
```
trainer = Trainer(
    model = model,
    args = args,
    train_dataset = lm_train,
    eval_dataset = lm_val if len(lm_val) > 0 else None,
    data_collator = collator,
)
```

[30]:
```
t0 = time.time()
trainer.train()
train_time_pt = time.time() - t0
print(f"Training (partial FT) took {train_time_pt/60:.2f}min")
```

```
/usr/local/lib/python3.12/dist-packages/torch/utils/checkpoint.py:85:
UserWarning: None of the inputs have requires_grad=True. Gradients will be None
  warnings.warn(
```

```
<IPython.core.display.HTML object>
```

```
Training (partial FT) took 0.31min
```

[31]:
```
# Perplexity on val
@torch.no_grad()
def compute_ppl_from_blocks(blocked_ds):
  total_loglik, total_tokens = 0.0, 0

  model.eval()
  for ex in blocked_ds:
    input_ids = torch.tensor(ex["input_ids"], device=model.device)

    out = model(input_ids = input_ids.unsqueeze(0),
                labels=input_ids.unsqueeze(0))

    total_loglik += out.loss.item() * input_ids.size(0)

    total_tokens += input_ids.size(0)

  return math.exp(total_loglik / total_tokens)
```

[32]:
```
model.to(device)
ppl_after_pt = compute_ppl_from_blocks(lm_val if len(lm_val) > 0 else lm_train)
print(f"Perplexity after partial fine-tuning: {ppl_after_pt:.2f}")
```

```
Perplexity after partial fine-tuning: 46.08
```

[33]:
```
GEN_PROMPT = (
     "Continue the following passage in the same satirical news style:\n\n"
    "Officials today announced a sweeping reform intended to fix a problem␣
 ↪nobody remembers creating."
)
inputs = tokenizer(GEN_PROMPT, return_tensors = "pt").to(device)

model.eval()
g0 = time.time()
gen_ids = model.generate(**inputs,
                         max_new_tokens = 150,
                         do_sample = True,
                         temperature = 0.85, # a puch to reduce rambling
                         top_p = 0.90,
                         repetition_penalty = 1.15, # stronger anti-looping
                         eos_token_id = tokenizer.eos_token_id,
                         pad_token_id = tokenizer.eos_token_id,
                         )
gen_time_pt = time.time() - g0
generated_pt = tokenizer.decode(gen_ids[0],
                                skip_special_tokens = True
                                )

print("\n=== Partial-FT Continuation (top-2 blocks) ===\n")
print(generated_pt[len(GEN_PROMPT):].strip())
print(f"\nGeneration (partial FT) took {gen_time_pt:.2f} s")
```

```
=== Partial-FT Continuation (top-2 blocks) ===

A pilot program aimed at improving air quality and avoiding high-fructose corn
syrup is on hold, despite criticism from some quarters that it has proved too
costly for any country's government or food safety advocates. The announcement
comes amid increasing pressure by Republican lawmakers over concerns about
possible "chronic health risks" associated with consuming refined sugar products
- including processed sugars like cane juice-and other beverages derived
directly out of raw crops such as potatoes, rice etc., which have been labeled
unsafe due largely because of their poor nutrition profile. Some experts warn
that artificial sweeteners are potentially toxic if not controlled thoroughly
before they can be sold across borders; others say even large quantities may
cause serious illness rather than cancer when consumed without proper labeling
(such use must also ensure good

Generation (partial FT) took 17.41 s
```

Re-evaluation of Partial Fine-Tuning

1. Qualitatively

- The generated continuation is grammatically fluent and coherent but still resembles policy/news reporting more than satire.
- The style lacks ironic exaggerations or witty punchlines, though sampling tweaks reduced repetition and rambling compared to full fine-tuning.
- The text tends to trail off rather than end with a sharp satirical closure.

2. Quantitatively

- Baseline perplexity (pre-tuning): 61.77
- Perplexity after full fine-tuning: 13.74
- Perplexity after partial fine-tuning: 46.08
- Interpretation: Partial fine-tuning improved over baseline but performed significantly worse than full fine-tuning in terms of token-level adaptation.

2.6 Discuss the trade-off between computational cost and sample generation performance based on the mothods you performed in this task.

Resource Tracking Report 1. Trainable Parameters - GPT-2 small total parameters: ~124M. - With embeddings + bottom 2 blocks frozen, most parameters still trainable. - Trainable parameters ~100M+ (majority of model). 2. Training Time - Training for 12 epochs took 0.24 minutes (~14 seconds) on GPU. - (Full fine-tuning reference: 0.36 minutes for 20 epochs). 3. Inference Time - Generating ~150 tokens took 2.35 seconds on GPU. - Comparable to full fine-tuning (~2.4 seconds).

Conclusion - - Partial fine-tuning reduced training time slightly but led to much higher perplexity than full fine-tuning. While fluency remained strong, stylistic alignment to satire was limited. The experiment shows that freezing only the bottom two layers is not sufficient for effective adaptation on this small dataset; methods like training only the top layers or LoRA adapters would likely be more efficient.

3 D Task

[36]:
```
from peft import LoraConfig, get_peft_model, TaskType

SEED = 276
set_seed(SEED)

device = "cuda" if torch.cuda.is_available() else "cpu"

MODEL_NAME = "gpt2"
OUTPUT_DIR_LORA = "out/gpt2-lora-satire"
```

[38]:
```
# base model
base_model = AutoModelForCausalLM.from_pretrained(MODEL_NAME)
base_model.resize_token_embeddings(len(tokenizer))
```

[38]:
```
Embedding(50257, 768)
```

3.1 Apply a parameter-efficient adaptation method (for example LoRA or adapter modules) to your model. Discuss the justification behind choosing an specific form of PEFT.

[41]:
```
# LoRA config
lora_cfg = LoraConfig(
    task_type = TaskType.CAUSAL_LM,
    r = 16,
    lora_alpha = 32,
    lora_dropout = 0.05,
    bias = "none",
    target_modules = [
        "c_attn", "c_proj", "c_fc", "c_proj"
    ])

lora_model = get_peft_model(base_model, lora_cfg)
lora_model.to(device)
```

```
/usr/local/lib/python3.12/dist-packages/peft/tuners/lora/layer.py:2174:
UserWarning: fan_in_fan_out is set to False but the target module is `Conv1D`.
Setting fan_in_fan_out to True.
  warnings.warn(
```

[41]:
```
PeftModelForCausalLM(
  (base_model): LoraModel(
    (model): GPT2LMHeadModel(
      (transformer): GPT2Model(
        (wte): Embedding(50257, 768)
        (wpe): Embedding(1024, 768)
        (drop): Dropout(p=0.1, inplace=False)
        (h): ModuleList(
          (0-11): 12 x GPT2Block(
            (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (attn): GPT2Attention(
              (c_attn): lora.Linear(
                (base_layer): Conv1D(nf=2304, nx=768)
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=768, out_features=16,
bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=16, out_features=2304,
bias=False)
                )
```

```
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
                (lora_magnitude_vector): ModuleDict()
              )
              (c_proj): lora.Linear(
                (base_layer): Conv1D(nf=768, nx=768)
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=768, out_features=16,
bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=16, out_features=768,
bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
                (lora_magnitude_vector): ModuleDict()
              )
              (attn_dropout): Dropout(p=0.1, inplace=False)
              (resid_dropout): Dropout(p=0.1, inplace=False)
            )
            (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (mlp): GPT2MLP(
              (c_fc): lora.Linear(
                (base_layer): Conv1D(nf=3072, nx=768)
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=768, out_features=16,
bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=16, out_features=3072,
bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
                (lora_magnitude_vector): ModuleDict()
              )
              (c_proj): lora.Linear(
                (base_layer): Conv1D(nf=768, nx=3072)
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.05, inplace=False)
```

```
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=3072, out_features=16,
bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=16, out_features=768,
bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
                (lora_magnitude_vector): ModuleDict()
              )
              (act): NewGELUActivation()
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
        (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      )
      (lm_head): Linear(in_features=768, out_features=50257, bias=False)
    )
  )
)
```

[42]:
```
# Showing ther trainable parameters
def count_params(m):
  tot = sum(p.numel() for p in m.parameters())
  trainable = sum(p.numel() for p in m.parameters() if p.requires_grad)
  return tot, trainable

total, trainable = count_params(lora_model)
print(f"Trainable parameters (LoRA): {trainable:,} / {total:,} ({100*trainable/
 ↪total:.2f}%)")
```

```
Trainable parameters (LoRA): 2,359,296 / 126,799,104 (1.86%)
```

[43]:
```
# a helper that prepares the dataset samples into proper batches for training
collator = DataCollatorForLanguageModeling(tokenizer = tokenizer, mlm = False)
```

3.2 Apply chosen method using your domain data. Generate same prompt continuation. Re-evaluate performance on sample generation. Compare style, fluency, and compare based on the evaluation metric you chose before.

[45]:
```
args_lora = TrainingArguments(
    output_dir = OUTPUT_DIR_LORA,
    per_device_train_batch_size = 1,
    per_device_eval_batch_size = 1,
    gradient_accumulation_steps = 16,
    num_train_epochs = 10,          # fewer than full FT
    learning_rate = 1e-4,           # slightly higher since fewer params
    weight_decay = 0.05,
    lr_scheduler_type = "cosine",
    warmup_ratio = 0.05,
    logging_steps = 10,
    save_steps = 100,
    save_total_limit = 2,
    eval_strategy = "no",     # keep simple; or use "steps" + eval_steps
    report_to = "none",
    fp16=torch.cuda.is_available() and not torch.cuda.is_bf16_supported(),
    bf16=torch.cuda.is_bf16_supported()
)

trainer_lora = Trainer(
    model = lora_model,
    args = args_lora,
    train_dataset = lm_train,
    eval_dataset = lm_val if len(lm_val) > 0 else None,
    data_collator = collator,
)
```

[47]:
```
t0 = time.time()
trainer_lora.train()
print(f"Training (LoRA) took {time.time() - t0:.2f}s")
```

```
<IPython.core.display.HTML object>
```

```
Training (LoRA) took 3.40s
```

[49]:
```
# Evaluation
lora_model.eval()

@torch.no_grad()
def perplexity_from_blocks(blocked_ds, model):
  total_ll, total_tok = 0.0, 0
  for ex in blocked_ds:
    ids = torch.tensor(ex["input_ids"], device = model.device)
    out = model(input_ids = ids.unsqueeze(0), labels = ids.unsqueeze(0))
```

```
    total_l1 += out.loss.item() * ids.size(0)
    total_tok += ids.size(0)
  return math.exp(total_l1 / total_tok)

ppl_lora = perplexity_from_blocks(lm_val if len(lm_val) > 0 else lm_train,␣
 ↪lora_model)
print(f"Perplexity after LoRA: {ppl_lora:.2f}")
```

```
Perplexity after LoRA: 38.89
```

[50]:
```
# GEneration from same prompt
gen_inputs = tokenizer(GEN_PROMPT, return_tensors = "pt").to(device)

with torch.no_grad():
  gen_ids = lora_model.generate(**gen_inputs,
                                max_new_tokens = 150,
                                do_sample = True,
                                temperature = 0.9,
                                top_p = 0.9,
                                repetition_penalty = 1.05,
                                eos_token_id = tokenizer.eos_token_id,
                                pad_token_id = tokenizer.eos_token_id,)

gen_text_lora = tokenizer.decode(gen_ids[0], skip_special_tokens = True)

print("\n=== LoRA Continuation ===\n")
print(gen_text_lora[len(GEN_PROMPT):].strip())
```

```
=== LoRA Continuation ===

Their first step is eliminating paycheque problems that could cause serious
financial mismanagement, and it would cut costs for hundreds of companies with
more than $200 million worth of assets at risk from their top management
decisions - such as employees who had worked hard enough but forgot or weren't
paying into retirement plans because they're too old for them. They plan on
cutting corporate taxes by 40 percent this year, including some state and local
governments.' The proposal goes even further than last week's announcement,
saying firms will also be able roll out new incentives like cash back awards
based entirely upon employee performance rather then individual achievements;
employers can apply these rewards automatically if executives win promotions
after five years' experience. "There are few better ways to reduce business
expenses," said
```

[ ]:
```

```